

REMARKS

Claims 1-16 and 27-52 are pending. Claims 17-26 have been canceled by a previous communication. By the present amendment, claim 1 has been amended to clarify this claim, and not in view of the prior art, and claims 48-52 have been added. For the reasons discussed in detail below, the rejections are traversed, and all of the pending claims are in condition for allowance.

The Office action has rejected claims 1-16 and 27-47 under 35 U.S.C. § 103(a) as being unpatentable over U.S. Patent No. 6,047,124 to Marsland (hereinafter Marsland) in view of U.S. Patent 5,491,808 to Geist Jr. (hereinafter Geist). Applicant respectfully traverses these rejections in the remarks below, which provides an overview of the invention and of the cited references, and then discusses the significant differences between the references and the claims.

In general, the present invention relates to testing kernel mode components, such as drivers loaded by an operating system, while those components are operating. Tests that correspond to specific types of errors to be monitored are identified to the kernel at the time of loading the component. To test the driver, when the loaded kernel component makes a system call, the call is re-vectorred to a special kernel mode verifier component that controls execution of the system calls, and establishes tests for the specific errors that were identified for that component. For example, if the type of error to be monitored was directed to memory-related errors, then the verifier component employs a series of memory monitoring techniques to watch for errors. Such memory monitoring techniques include isolating the memory allocated to a component to see if the component writes to memory locations beyond that which was allocated, testing memory usage, and marking deallocated memory. The verifier may also check whether all allocated memory is deallocated when the component is unloaded, to detect memory leaks. Additionally, the verifier may check whether any requested kernel resources were deleted when the component unloads.

These resources may include queues, lookaside lists, worker threads, timers and other resources. To increase the chances of finding errors that only occur in certain conditions such as when available memory is low, the component verifier also may to simulate low resource conditions to discover improper component behavior at such times. Note that the above description is for informational purposes only, and should not be used to interpret the claims, which are discussed below.

In contrast to the present invention, Marsland is generally related to a passive tracing device. Specifically, Marsland describes tracing signals and calls. The tracing driver is interposed between the application's user mode process and the traced device driver, to trace interactions occurring between the traced device driver and the application process and the kernel. In general, Marsland describes tracing activities such as observing incoming and outgoing system calls, identifying the applications using device drivers, and time stamping events and tracking resources allocated and used by a device driver. Marsland describes a tracing device driver that operates in kernel memory space, which works with a tracing process that operates in the user memory space. The tracing process controls the tracing device driver, and has a user interface for setting parameters of the tracing device driver and displaying tracked activity.

Geist relates to tracking user mode memory allocation and deallocation for an application program in a network file server. The technique uses an allocation tracker to record memory allocation by user programs and exception messages for memory requests. Geist describes an application monitor that operates in user memory space which attempts to overcome the limitation of netware loaded modules (NLM) that do not identify individual memory allocations, deallocations, and reallocations as they occur in the running of a particular NLM. The network operating system makes a set of APIs available to NLM applications to request memory services.

Geist's technique intercepts memory allocation calls made by a user program via the C library to the kernel, and logs the allocations and releases resulting from those application calls. The tracker logs the allocations and releases using two simple linked lists: an allocated block list and a message list. The technique tracks allocations without release, and release requests for unallocated memory.

The tracker works by modifying the entry points of the APIs for memory allocation routines in the C library. A jump instruction is placed at the beginning of the allocation routines to an assembly language routine that calls the rest of the original routine and then calls a stub at the end to log allocation or deallocation requests that are made. Geist's allocation tracker provides reports to give developers of application programs an opportunity find memory allocation problems in their application programs.

The following table lists the elements of claim 1 and the cited prior art sections upon which the Office action has relied as allegedly disclosing those elements.

	Claim 1 Element	Prior Art
A.	receiving a request from a kernel mode driver	Moreover, the tracing driver allows the originator of the request to be identified, including a specific user, process, thread or a request pertaining to a specific logical device provided by that driver. (Marsland, 5: 45-49)
B.	determining that the driver is to be monitored	Operationally, the tracing driver enables a user to specify which device drivers and what events are to be traced. During subsequent processing, the kernel records the traced events for later capture and dissemination to other user processes. ... (Marsland, 5: 38-67)
C.	re-vectoring the request to a kernel mode driver verifier	... the tracker NLM takes over the memory related operating system calls ... by finding the actual memory management functions of each other server resident NLM by examining their global symbol lists and replacing the first few instructions (called the "prefix") of each function with an instruction to cause a jump (JMP) to a special assembly routine, or "thunk." ... (Geist, 7:14-62)

D.	taking action in the driver verifier to actively test the driver for errors	Fig. 5 is a flow diagram of the routine for performing a driver trace 63 with the dtrace process 52 used by the method of Fig 4. By way of example, the routine identifies and processes three types of events, although additional events could also be traced. Thus if an entry point execution event is identified (block 70), the entry point execution is processed (block 71) as further described herein below with reference to Fig. 6. ... (Marsland, 6: 3-14)
----	---	--

Element C of claim 1 recites “re-vectoring the request to a kernel mode driver verifier.”

The Office action (on page 2 thereof) has conceded that Marsland does not teach the recited limitation, and has relied on Geist to allege that the recited limitation would be obvious in view of Marsland. This is incorrect.

Whenever a particular driver is to be monitored in the present invention, a kernel system call received from that driver is re-vectorred to the driver verifier. The driver verifier invokes a kernel test function in place of the requested kernel function. In contrast to Geist, in the present invention, both the driver and the driver verifier are kernel components executing in the kernel address space. Geist does not re-vector calls in the kernel mode, but instead replaces the C library memory function calls in the application executable with a jump instruction to a routine that records memory allocations and deallocations, along with executing the original API that invokes the requested memory function. *Geist*, column 7, lines 47-62.

Such C library functions are linked and loaded in the user address space. The routine that Geist describes for recording memory allocations and deallocations is also part of the application executable loaded in user address space. Only when the C library function calls the kernel function, passing parameters, is protected kernel space executed, with results placed in the user

space for the user application to retrieve upon resuming execution. In fact, in Geist's kernel, nothing is changed, let alone re-vectored.

The modified C library function of Geist executes in user space and does not have direct access to kernel routines. It would not have been obvious, as alleged, that Marsland would use this technique to somehow trace inter-device driver calls. In fact, Marsland specifically *teaches away* from such an approach because it is limited to user mode involvement. More particularly, in rejecting a prior art technique of interposing a layer between applications in user memory space and kernel components in kernel memory space, Marsland specifically states that such a technique is limited to tracing system call interactions occurring across the user and kernel memory space boundary. Marsland, column 1, lines 52-55. Marsland explains that device driver interfaces execute within the kernel memory space and are not accessible by application programs executing in user memory space, and no suggestion is made to somehow modify the application programming interfaces between the kernel mode components and the user mode application program code for accessing inter-kernel communications.

In contrast to Marsland and/or Geist, claim 1 applicant's technique instead provides a driver verifier that may include separate kernel functions to replace or perform some additional functions with respect to existing kernel functions via re-revectoring. Applicant's substitute and/or enhanced kernel functions execute in kernel memory space, and thus have access to kernel routines. This is significantly different from Marsland's teachings, and entirely different from Geist's teachings (which do not affect the kernel).

Further, element D (of claim 1) in the table above recites "taking action in the driver verifier to actively test the driver for errors." Applicant's driver verifier is not a passive tracer as described in Marsland or a passive tracker as described in Geist. The driver verifier actively tests the driver,

e.g., as the verifier performs automatic verification of a device driver by applying test conditions designed to detect specific errors during controlled operation of the executing driver. For example, applicant's driver verifier can specifically test for the driver improperly accessing memory locations outside of those memory locations allocated to the driver. This may be accomplished by allocating the driver's memory from a special pool (e.g., claim 4) owned by the driver verifier which is explicitly bounded by inaccessible memory space. Any writes outside the bounded area generate an error at that moment, which isolates the error condition within the context of the current machine state, rather than simply tracing the problem for later possible use.

Another active test that the driver verifier may perform is a test for a driver improperly accessing memory that the driver has already deallocated. The driver verifier does this by marking deallocated memory from the verifier's special pool as inaccessible, at the time that memory is deallocated (e.g., claim 6). Any attempt to again access this deallocated memory generates an error. The driver verifier also can perform other active tests.

In contrast, Marsland's system does not support active testing because the tracing driver is a layer of code interposed between the application process and device driver for recording transactions performed by the kernel on that device driver. It is limited to forwarding requests onto the underlying device driver in untranslated form. It only has hooks for recording runtime statistics of driver and resource usage. Moreover, in addition to not actively testing for memory access violations or leakage while the driver is running, Marsland's tracer does not, for example, examine resource lists at driver unload (e.g., claim 12).

To establish *prima facie* obviousness of a claimed invention, all of the claim limitations must be taught or suggested by the prior art; (*In re Royka*, 490 F.2d 981, 180 USPQ 580 (CCPA 1974)), and "all words in a claim must be considered in judging the patentability of that claim

against the prior art;" (*In re Wilson*, 424 F.2d 1382, 1385, 165 USPQ 494, 496 (CCPA 1970)).

Further, if prior art, in any material respect teaches away from the claimed invention, the art cannot be used to support an obviousness rejection. *In re Geisler*, 116 F.3d 1465, 1471, 43 USPQ2d 1362, 1366 (Fed Cir. 1997).

The present Office action has failed each of these requirements with respect to claim 1 for at least the reasons set forth above, e.g., Geist's modifying of a jump instruction in user mode code so as to temporarily transfer control to a user mode logging mechanism is not re-vectoring a call to a kernel mode verifier, and in any event is taught away from by Marsland; and the tracing mechanism of Marsland is a passive recording mechanism, not an active testing mechanism.

Claims 2-16, by similar analysis, are not rendered obvious by the relied-upon references.

Reconsideration and withdrawal of the rejections based on Marsland and Geist are respectfully requested.

Turning to the rejections of claims 27-30, applicant respectfully submits that claims 27-30 are not obvious as alleged, and that the relied-upon sections of the prior art used to allege obviousness were incorrectly interpreted. For example, claim 27 recites "restricting access to areas bounding the location" of allocated memory and "monitoring the areas bounding the location for an access violation." Marsland and Geist simply do not teach such subject matter, neither explicitly nor implicitly as alleged. Indeed, the cited section of Geist describes a standard "malloc" API used by an application program to request an allocation of bytes of memory. Geist merely describes that "malloc" returns a pointer to an allocated block of memory. In contrast to the Office action's contention, Geist in no way further teaches or suggests that this block of memory returned would be bounded by memory areas of restricted access.

In making the rejection, the Office action appears to be misinterpreting both plain the claim language and the way in which memory access works in the kernel mode. Claim 27 does not recite that the allocated memory is restricted (to the driver that requested allocation), but rather that access to the areas of memory *bounding* the allocated location are restricted, i.e., “restricting access to areas bounding the location.” In other words, the Office action has it entirely backwards; the areas bounding the location are restricted so that a tested driver, for example, *cannot* access the bounded areas. This is not restricting the space so as to only being accessible by that driver, as the Office action has essentially concluded; instead, no driver, including the tested one, can access the restricted bounding areas without causing an access violation. The rejection is based on a basic misreading of the plain claim language.

Further, the rejection of this claim is based on a faulty premise, namely that allocated memory is restricted to the entity that requested it. This is (normally) true with user mode memory as in Geist’s user mode allocations, because the operating system enforces access to only that user mode process to which the memory was allocated. However, this is not true when dealing with kernel memory. As generally described in the background section of applicants’ specification, for various performance and architectural reasons, any kernel component can access another kernel component’s allocated memory, although this is (ordinarily) an error. The driver verifier tests for such errors (and possibly others) via the method recited in claim 27, while Marsland and Geist do no such thing.

For at least the foregoing reasons, claims 27-30 are patentable over the prior art of record. Reconsideration and withdrawal of these rejections are respectfully requested.

Applicant respectfully submits that claim 31 is also not rendered obvious by Marsland and Geist, and that the cited sections were again misinterpreted with respect to the plain claim

language. Claim 31 includes the limitations of “receiving a plurality of requests from a component for allocation of various distinct sets memory” and “determining from the tracking whether space remains allocated to the driver at a time when the driver should have no space allocated thereto.” Applicant’s technique actively detects when a driver unloads without deallocating its memory, for example.

Geist does not describe such subject matter, despite the allegation in the Office action. Instead, the cited section of Geist describes use of an application tracker to record where unfreed memory of an application was originally allocated. Geist may generate a report for a developer to later review, but does not disclose or in any way suggest that the application tracker is able to determine whether space remains allocated to a driver at a time when the driver should have no space allocated thereto, as claimed, let alone provide any explanation as to how such a determination may be made, or why it might be desirable to do so. Further, unlike Geist’s application programs, drivers are kernel components executing in the kernel address space and are not accessible by application programs executing in user memory space. For at least the foregoing reasons, claim 31 is patentable over the prior art of record. Reconsideration and withdrawal of the rejection of claim 31 is respectfully requested.

Applicant respectfully submits that claims 32-34 are likewise not rendered obvious by Marsland and Geist. Claim 32 recites “receiving information corresponding to a driver unload,” “determining whether resources remain associated with the driver,” and “if so, generating an error.” For example, these resources may include queues, lookaside lists, worker threads, timers and other resources. Neither Marsland nor Geist describe or suggest detecting unreleased kernel resources and generating an error upon driver unload, in contrast to the plain claim language of claim 32.

Instead, Geist describes a passive application tracker that operates in user memory space and (needless to say) cannot access kernel components such as a driver that executes in kernel memory space. Geist's tracker logs the allocations and releases for application programs using two simple linked lists: an allocated block list and a message list. The cited section of Geist describes the data structure for the message list that records an error code for application program errors. Geist does not describe generating errors, let alone for kernel resources that are unreleased by device drivers, nor suggest any mechanism that might possibly accomplish this. Claims 33-34, by similar analysis, are not rendered obvious by the relied-upon references. Reconsideration and withdrawal of the rejections of claims 32-34 are respectfully requested.

Applicant respectfully submits that claims 35-47 (as well as newly added claims 48-52) are not rendered obvious by Marsland and Geist. Claims 35-47 include the limitations of a "re-vectoring component" and a "driver verifier component." As discussed above with respect to claim 1, as conceded in the Office action, Marsland does not teach re-vectoring a request to a driver verifier, and Geist also fails to disclose such an element. Neither Marsland's passive tracer nor Geist's passive tracker provide testing along the lines of the claimed driver verifier component. Reconsideration and withdrawal of the rejection of claims 35-47 are respectfully requested.

Moreover, the combination of Marsland and Geist is impermissible as a matter of law. To guard against the use of impermissible hindsight based on applicants' teachings, it is well settled that it is not permissible to combine references absent some teaching, suggestion, or motivation to combine the references. See, e.g., *In re Fine*, 837 F.2d 1071, 5 USPQ2d 1596 (Fed. Cir. 1988); *In re Jones*, 958 F.2d 347, 21 USPQ2d 1941 (Fed. Cir. 1992); *In re Geiger*, 815 F.2d 686, 688, 2 USPQ2d 1276, 1278 (Fed. Cir. 1987).

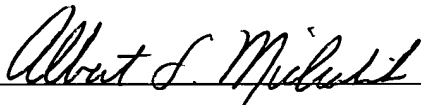
In the present case, it appears that the Office action can only have used impermissible hindsight based on applicants' own teachings to locate one reference (Marsland) that is a passive tracking mechanism, and another passive tracking mechanism (Geist, which operates in user mode in a manner that Marsland specifically teaches is unworkable for kernel mode drivers), in a (failed) attempt to reconstruct applicants' claims in order to allege obviousness. This can only have been done using applicants' teachings as the motivation for making the combination, and as a blueprint in the failed attempts to piece together the prior art into the claim limitations. However, such a hindsight reconstruction based on and motivated by applicants' teachings is impermissible by law, and for at least this additional reason, the claims are patentable over the prior art of record.

CONCLUSION

For at least the foregoing reasons, the claims are patentable over the prior art of record. Reconsideration and withdrawal of the rejections and timely allowance of this application is respectfully submitted.

If in the opinion of the Examiner a telephone conference would expedite the prosecution of the subject application, the Examiner is invited to call the undersigned attorney at (425) 836-3030.

Respectfully submitted,



Albert S. Michalik, Reg. No. 37,395
Attorney for Applicants
Law Offices of Albert S. Michalik, PLLC
704 - 228th Avenue NE, Suite 193
Sammamish, WA 98074
(425) 836-3030
(425) 836-8957 (facsimile)